Algoritmos e Estruturas de Dados MEEC

Tabelas de Dispersão

Tabelas de Dispersão - Introdução (1)

- As tabelas de dispersão (hash tables) são estruturas de dados adequadas para:
  - Tabelas de símbolos para compiladores
  - Dicionários
  - etc.

Elementos identificados por uma chave genérica

- As tabelas de dispersão beneficiam da elevada rapidez revelada pelas tabelas/listas nas principais operações de manipulação de dados:
  - Inserção (rápida em listas)
  - Acesso (rápida em tabelas)

Tabelas de Dispersão - Introdução (2)

- Relembrando a eficiência das operações em várias alternativas de representação:

	Listas	Árvores	Tabelas
Acesso	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
Inserção	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(N)$
Procura	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$

- A chave evita procura (sempre custosa)
- Será possível usar capacidade ilimitada da lista, com a eficiência de uma tabela? Se sim, como inserir um dado numa tabela cheia?

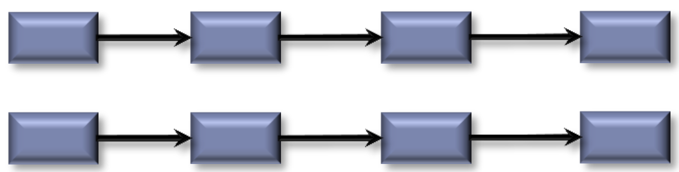
Chave: campo único em cada registo (ex: número de aluno)

Tabelas de Dispersão - Introdução (3)

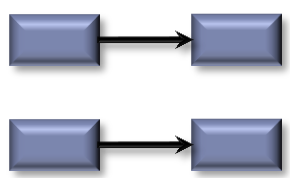
- Dispersão de 8 elementos por *N* listas (*N*=1,2,3,...).

\- 1 lista:

\- 2 listas:



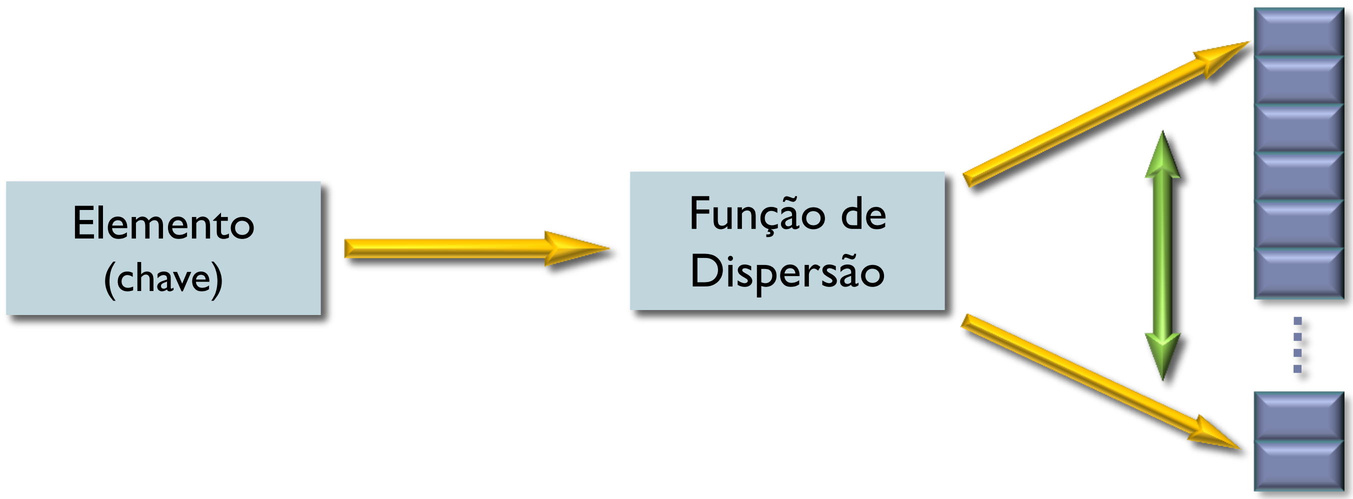
\- 3 listas:



Conclusão: é necessária uma função de distribuição de elementos por cada lista.

Tabelas de Dispersão - Introdução (4)

- No limite, teremos uma tabela de *N* posições
  - Em cada posição temos uma lista de tamanho 1.
- Função de dispersão
  - Converte a chave genérica de cada elemento num índice da tabela.



Tabelas de Dispersão - Introdução (5)

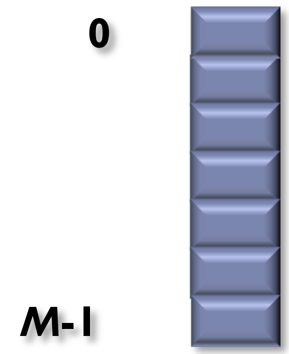
- Colisão: situação em que duas chaves diferentes resultaram, pela função de dispersão, no mesmo índice da tabela.
- É necessário estabelecer a estratégia para resolução de colisões
- Compromisso entre espaço-tempo
  - Sem limitações de espaço:
    - função de dispersão trivial, em que a chave resulta num endereço de memória.
  - Sem limitações de tempo:
    - resolução de colisões através de procura sequencial.
  - Com limitações de espaço e tempo:
    - Tabelas de Dispersão

Tabelas de Dispersão – Introdução(6)

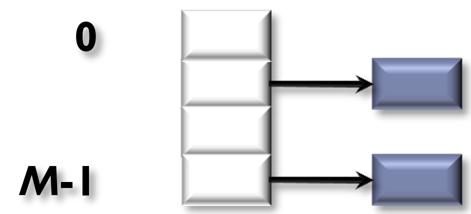
- As tabelas de dispersão utilizam:
  - tabela base de indexação
  - funções de dispersão
  - algoritmos para resolução de colisões (se necessário)

Tabela Base

- Métodos de dispersão com índices livres:
  - Quando o número de elementos pode ser estimado à partida em *N*
  - Tabela de *M* elementos (*M* > *N*)



- Métodos de dispersão por separação em listas:
  - Quando o número de elementos a guardar (*N*) é desconhecido
  - Tabela de M listas de elementos (*M* < *N*)



Funções de Dispersão (1)

- Objectivo: Distribuir uniformemente chaves do elementos por inteiros entre [0, *M*-1] (evitar colisões)

- A função de dispersão mais usada é a **divisão**

- Chaves pequenas (representáveis por uma palavra de memória):
  - tratar as chaves como inteiros (*k*)

    $h(k) = k \% M$ (em C, % calcula o resto da divisão inteira)
  - usar um número primo para *M*

Funções de Dispersão (2)

Chave k	k%1000	k%1024	k%1021
32699	**699**	**955**	027
15114	114	778	820
41502	502	542	662
30444	444	748	835
81699	**699**	803	019
30651	651	**955**	021
23670	670	118	187
12219	219	**955**	988
75745	745	993	191

1 colisão, usados apenas os 3 últimos dígitos

2 colisões, usados apenas os 10 bits de menor peso

Funções de Dispersão (3)

- Chaves longas (cadeias de caracteres)
  - Usar cada caracter como um inteiro (valor da representação ASCII)

$$h(k) = k \% M$$

  - Atribuir um peso a cada caracter correspondente à sua posição na cadeia
  - Usar um número primo para o tamanho da tabela

```
#define PESO 117
int hash(char *s, int M)
{
   unsigned h;
   for (h=0; *s!=´\0´; s++)
      h = (PESO*h + (unsigned)*s) % M;
   return h;
}
```

Caracter	ASCII
a	97
b	98
c	99

`hash("abc",101) = 37`

`PESO`, número primo da ordem de *M*, usado normalmente quando se tem *M*>>**`strlen(s)`**

Funções de Dispersão (4)

- Outras funções (computacionalmente mais complexas)
  - Mediania: somar os quadrados dos valores obtidos em cada símbolo, retirando n bits do meio do código dos caracteres ($n = \log_2 M$)
  - Partição: dividir o código dos caracteres em segmentos de n bits, somar os segmentos e tomar os n bits de menor peso

Resolução de Colisões (1)

- Para uma tabela de tamanho M, quantas inserções podem ser feitas até à primeira colisão?
  - Problema clássico de probabilidades:
    - Para uma função de dispersão "aleatória" as primeiras colisões ocorrem ao fim de sqrt(π*M/2)

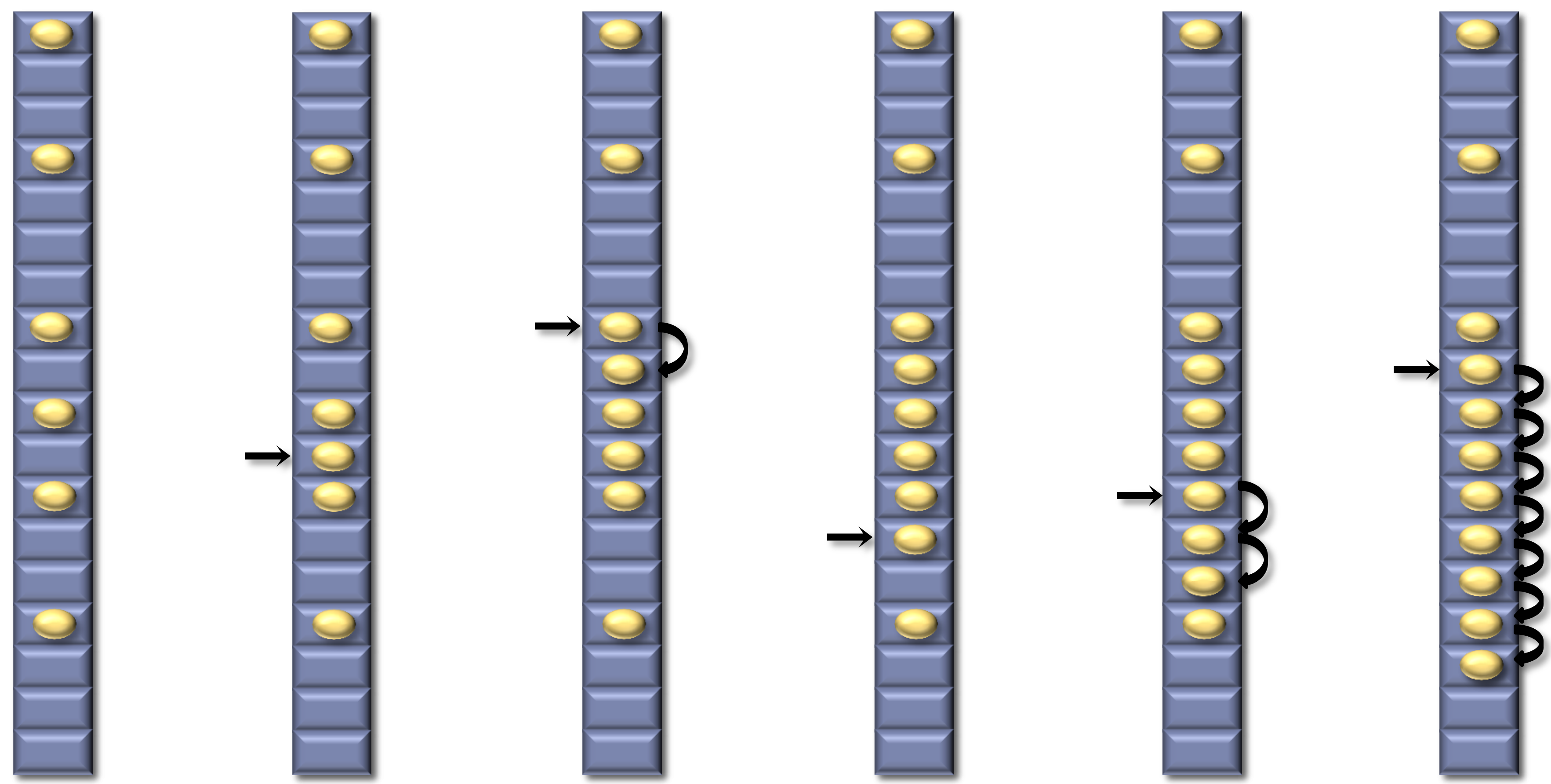
M	sqrt(π*M/2)
100	12
1021	40
10000	125

- Os algoritmos de resolução de colisões depende do tipo de tabela base escolhido

Resolução de Colisões
Procura linear (2)

- $N < M$: método com índices livres.
- Dado que há sempre posições livres na tabela, procurar outra posição.
- Procura linear
  - Se a posição correspondente ao índice devolvido pela função de dispersão estiver ocupada, ir incrementando o índice até se encontrar uma posição livre.

Resolução de Colisões
Procura linear (3)



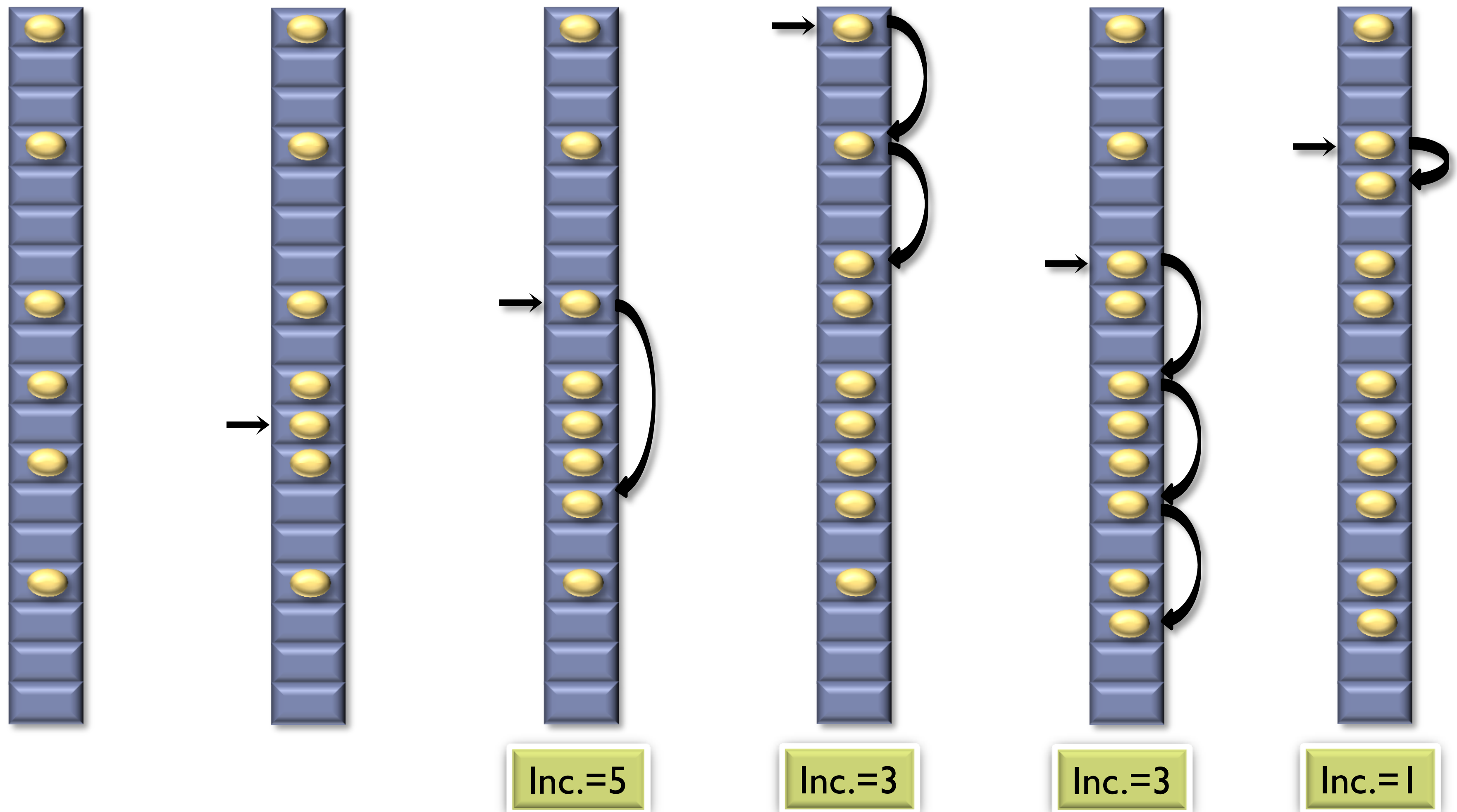
Resolução de Colisões
Procura linear (4)

- Desempenho:
  - Os elementos tendem a ficar agrupados (clusters)
  - Os agrupamentos grandes tendem a crescer ainda mais
  - O tempo médio de procura tende a crescer para M à medida que a tabela enche
  - Operações na tabela de dispersão tornam-se demasiado lentas quando a tabela atinge 70% a 80% da sua capacidade.

Resolução de Colisões
Dupla Dispersão (5)

- Dupla dispersão:
  - Se a posição correspondente ao índice devolvido pela função de dispersão estiver ocupada, utilizar outra função de dispersão para determinar o valor a incrementar na procura de uma posição livre para o elemento
    - aleatória
    - quadrática ($p_i=ai^2+bi+c$, em que a,b,c são função da máquina)
    - linear ($p_i=ni$)
- Desempenho:
  - Evita a criação de agrupamentos
  - Operações na tabela de dispersão só se tornam demasiado lentas quando a tabela atinge 90% a 95% da sua capacidade

Resolução de Colisões
Dupla Dispersão (6)



Resolução de Colisões (7)

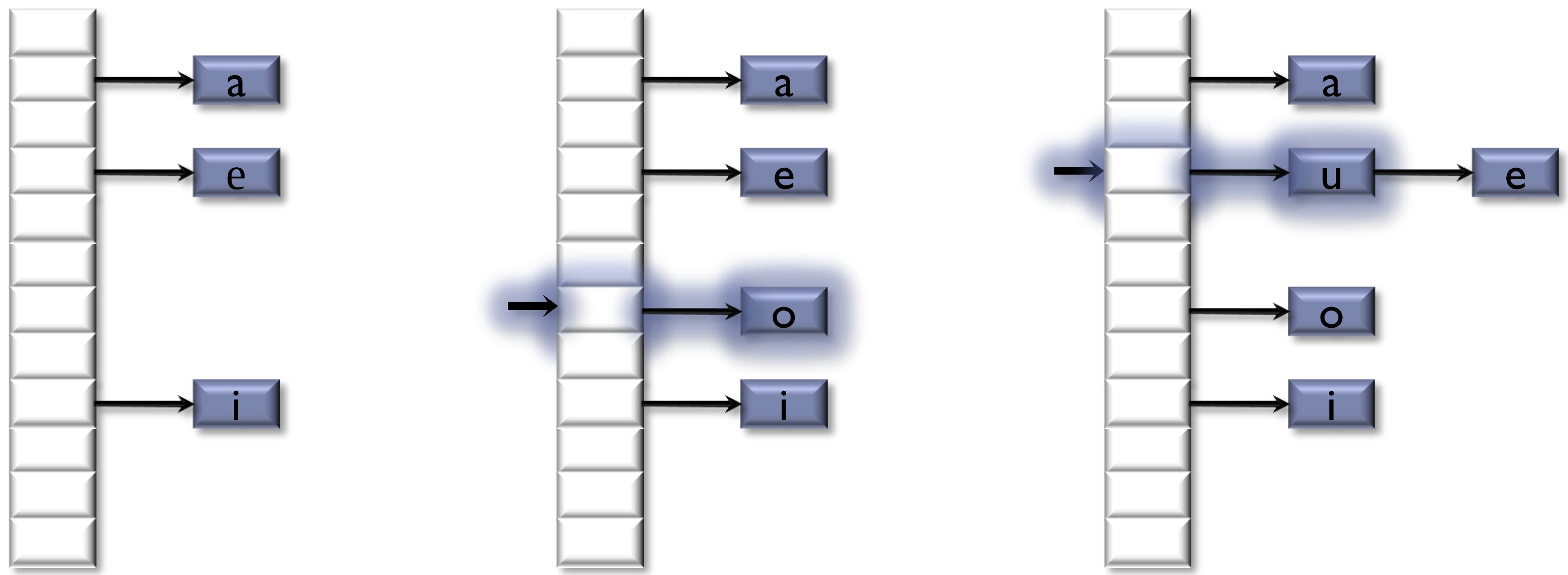
- Número médio de comparações em função do factor de carga l_f (load factor) = N/M:
  - Aleatório
    - $E_r = -1/l_f * \ln(1 - l_f)$
  - Linear
    - $E_l = (1 - l_f/2)/(1 - l_f)$
  - Encadeado
    - $E_c = 1 + l_f/2$

l_f	E_l	E_r	E_c
20%	1.13	1.12	1.10
50%	1.50	1.39	1.25
80%	3.00	2.01	1.40
90%	5.50	2.56	1.45
95%	10.50	3.15	1.48

Resolução de Colisões
Inserção no início (8)

- *N* > *M*: método de separação em listas (endereçamento encadeado).
- Dado que há menos posições na tabela que elementos, vão de certeza ocorrer colisões.
- Inserção no início
  - Se a lista correspondente ao índice devolvido pela função de dispersão já contiver elementos, inserir o novo elemento no início da lista.

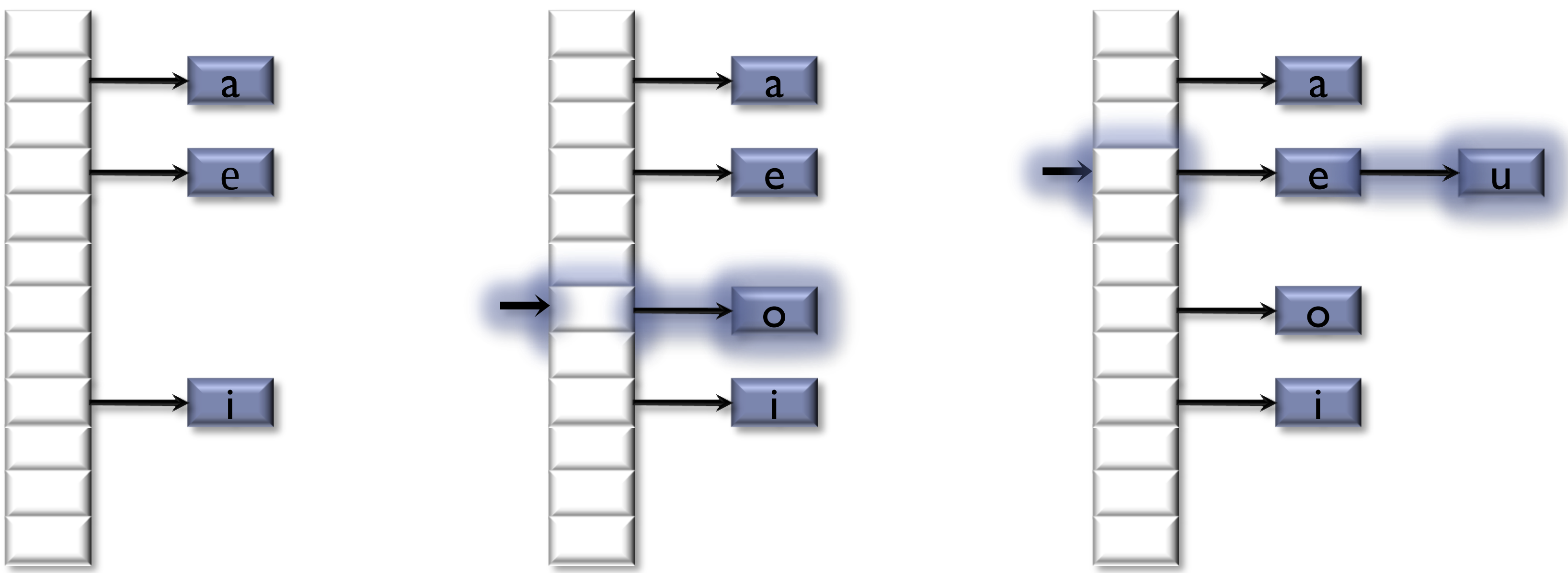
Resolução de Colisões
Inserção no início (9)



- Custo de inserção: 1
- Custo médio de procura (sem sucesso): N/M
- Custo médio de procura (com sucesso): $N/(2M)$

Resolução de Colisões
Inserção ordenada (10)

- <u>Inserção ordenada</u>:
  - se a lista correspondente ao índice devolvido pela função de dispersão já contiver elementos, inserir ordenadamente o novo elemento na lista.



- Custo de inserção: $N/(2M)$
- Custo médio de procura (com ou sem sucesso): $N/(2M)$

Resolução de Colisões (11)

```c
#include <stdlib.h>
#include <string.h>
#define HASHSIZE 211

typedef struct _s {
   char *name;  int id;
   struct _s *next;
} table_element;

table_element *HashTab[HASHSIZE];

table_element *search(char *Str)
{
   table_element *ptr = HashTab[hash(Str, HASHSIZE)];
   for(  ;  ptr != NULL; ptr = ptr->next)
      if(!strcmp(Str, ptr->name)) return ptr;
   return NULL;
}
```

Resolução de Colisões (12)

```
table_element *insert(char *Str, int Val)
{
   table_element *ptr;
   unsigned HashVal;

   if((ptr = search(Str)) == NULL){
      ptr = (table_element *)malloc(sizeof(table_element));
      if(ptr == NULL) return NULL;

      HashVal = hash(Str, HASHSIZE);
      ptr->name = strdup(Str);
      ptr->id = Val;
      ptr->next = HashTab[HashVal];
      HashTab[HashVal] = ptr;
   }
   return ptr;
}
```

Tabelas de Dispersão - Conclusão

- Implementam operações de inserção e procura em tempo constante (análise amortizada).
- Desvantagens:
  - Não há garantia de desempenho ("performance")
  - Chaves longas podem envolver muita aritmética
  - Podem ocupar mais memória do que necessário
  - Não suportam eficientemente todas as operações sobre tipos de dados abstractos (ordenação, por exemplo)

Síntese da Aula 1 de Tabelas de Dispersão

- Introdução às tabelas de dispersão
- Componentes das tabelas de dispersão
- Funções de dispersão
  - divisão
- Resolução de colisões
  - Dupla dispersão
  - Encadeado
  - Comparação da eficiência